26장. ES6 함수의 추가 기능

26.1 함수의 구분

1. ES6 이전까지 자바스크립트의 함수는 별다른 구분 없이
   - 일반적인 함수로서
   - new 연산자와 함께 생성자 함수로서
   - 객체에 바인딩되어 메서드로서다양한 목적으로 사용되었다.

   → 편리한 것 같지만 **실수를 유발, 성능 면에서도 손해**

2. [예제 26-01]

```
var foo = function () {
    return 1;
};

// 일반적인 함수로서 호출
foo();  // 1

// 생성자 함수로서 호출
new foo();  // foo {}

// 메서드로서 호출
var obj = { foo: foo };
obj.foo();  // 1
```

   - 이처럼 사용 목적에 따라 명확히 구분 X

   ▶ 즉, ES6 이전의 모든 함수는 일반 함수로서 호출할 수 있고, → callable

   생성자 함수로서 호출할 수 있다. → constructor

3. ES6 이전에 일반적으로 **메서드**라고 부르던 함수도 callable & constructor

4. 이처럼 ES6 이전의 모든 함수는 사용 목적에 따라 명확히 구분 X, 호출 방식에 특별한 제약 X, 생성자 함수로 호출되지 않아도 프로토타입 객체를 생성

→ **혼란, 실수 유발 가능성, 성능에도 좋지 않음**

5. 이러한 문제를 해결하기 위해 ES6에서는 함수를 사용 목적에 따라 세 가지 종류로 명확히 구분

ES6 함수의 구분	constructor	prototype	super	arguments
일반 함수	O	O	X	O
메서드	X	X	O	O
화살표 함수	X	X	X	X

26.2 메서드

1. **ES6 사양에서 메서드는 메서드 축약 표현으로 정의된 함수만을 의미**

2. **ES6 사양에서 정의한 메서드(이하 ES6 메서드)는 인스턴스를 생성할 수 없는 non-constructor**
   + 인스턴스를 생성할 수 없으므로, prototype 프로퍼티가 없고 프로토타입도 생성 X

3. **ES6 메서드는 자신을 바인딩한 객체를 가리키는 내부 슬롯 [[HomeObject]]를 갖는다.**
   [[HomeObject]]를 갖는 ES6 메서드는 super 키워드 사용 가능

26.3 화살표 함수

1. 화살표 함수(arrow function)는 표현만 간략한 것이 아니라, 내부 동작도 기존 함수보다 간략.

2. 특히 콜백 함수 내부에서 this가 전역 객체를 가리키는 문제를 해결하기 위한 대안으로 유용

26.3.1 화살표 함수 정의

함수 정의

- 함수 선언문으로 정의할 수 없고, 함수 표현식으로 정의해야 함.

매개변수 선언

- 매개변수가 여러 개인 경우 소괄호 ( ) 안에 매개변수 선언

- 매개변수가 한 개인 경우 소괄호 ( ) 생략 가능
- 매개변수가 없는 경우 소괄호 ( ) 생략 X

함수 몸체 정의

- 함수 몸체가 하나의 문으로 구성된다면 중괄호 { }를 생략할 수 있다.
- 함수 몸체 내부의 문이 표현식이 아닌 문이라면 에러 발생
- 객체 리터럴 반환하는 경우, 객체 리터럴을 소괄호 ( )로 감싸 주어야 함.
- 즉시 실행 함수(IIFE)로 사용 가능

➡️ **이처럼 화살표 함수는 콜백 함수로서 정의할 때 유용**

26.3.2 화살표 함수와 일반 함수의 차이

1. **화살표 함수는 인스턴스를 생성할 수 없는 non-constructor다.**

```
const Foo = () => {};
// 화살표 함수는 생성자 함수로서 호출할 수 없다.
new Foo(); // TypeError:~
```

2. **중복된 매개변수 이름을 선언할 수 없다.**

```
const arrow = (a,a) => a + a;
// SyntaxError:
```

3. **화살표 함수는 함수 자체의 this, arguments, super, new.target 바인딩을 갖지 않는다.**
    - 따라서 화살표 함수 내부에서 this, arguments, super, new.target을 참조하면 스코프 체인을 통해 상위 스코프의 this, arguments, super, new.target을 참조한다.

26.3.3 this

1. 화살표 함수가 일반 함수와 구별되는 가장 큰 특징은 바로 this.
    - **“콜백 함수 내부의 this 문제”**

      콜백 함수 내부의 this가 외부 함수의 this와 다르기 때문에 발생하는 **문제**

→ 이를 해결하기 위해 의도적으로 설계된 것

2. 주의할 것❗ : 일반 함수로서 호출되는 콜백 함수의 경우

   [예제 26-28]

```
class Prefixer {
  constructor(prefix) {
    this.prefix = prefix;
  }

  add(arr) {
      // 1번
    return arr.map(function(item) {
      return this.prefix + item;  // 2번
      // TypeError 에러 발생
    });
  }
}

const prefixer = new Prefixer('-webkit-');
console.log(prefixer.add(['transition', 'user-select']));
```

   - 클래스 내부 모든 코드에는 strict mode

     따라서 Array.prototype.map 메서드의 콜백 함수에도 strict mode

     ⇒ 2번 this에는 undefined가 바인딩

   - 1번 this ≠ 2번 this : 서로 다른 값을 가리키고 있음

3. **화살표 함수는 함수 자체의 this 바인딩을 갖지 않는다.**

   따라서 화살표 함수 내부에서 this를 참조하면 상위 스코프 this를 그대로 참조

   → 이를 **lexical this**라 함.

4. 만약 화살표 함수, 화살표 함수가 중첩되어 있다면 상위 화살표 함수에도 this 바인딩 X

   → 스코프 체인 상 가장 가까운 상위 함수 중에서 화살표 함수 아닌 함수의 this를 참조

5. 화살표 함수로 메서드를 정의하는 것은 바람직하지 않다.

   = 메서드 정의할 때는 ES6 메서드를 사용하는 것이 좋다.

26.3.4 super

1. this와 마찬가지로 상위 스코프의 super를 참조.

26.3.5 arguments

1. arguments 객체 : 함수 정의할 때 매개변수 개수를 확정할 수 없는 가변 인자 함수 구현시 유용

2. 화살표 함수에서는 사용할 수 없다 ⇒ 반드시 Rest 파라미터 사용

26.4 Rest 파라미터

26.4.1 기본 문법

1. **Rest 파라미터는 함수에 전달된 인수들의 목록을 배열로 전달받는다.**

   [예제 26-49]

```
function foo(...rest) {
    console.log(rest);    // [ 1, 2, 3, 4, 5 ]
}

foo(1, 2, 3, 4, 5);
```

2. Rest 파라미터는 이름 그대로 먼저 선언된 매개변수에 할당된 인수를 제외한,
   나머지 인수들로 구성된 배열이 할당된다. 따라서, Rest 파라미터는 반드시 마지막이어야 함.

   [예제 26-51]

```
function foo(...rest, param1, param2) { }

foo(1, 2, 3, 4, 5);
// SyntaxError
```

3. Rest 파라미터는 단 하나만 선언할 수 있다.

4. length 프로퍼티에 영향을 주지 않는다.

26.4.2 Rest 파라미터와 arguments 객체

1. arguments 객체는 배열이 아닌 유사 배열 객체이므로 배열 메서드 사용 시 번거로움

   ➡️ ES6에서는 rest 파라미터를 사용하여 번거로움을 피할 수 있다.

26.5 매개변수 기본값

1. 자바스크립트 엔진은 매개변수 개수, 인수 개수를 체크하지 않음

   → 개수를 제대로 전달 안해도 에러 발생 X, 허나 **방치하지 않는 것이 좋다.**

   - 인수가 전달되지 않은 매개변수의 값은 undefined.

2. 인수가 전달되지 않은 경우 매개변수에 기본 값을 할당할 필요가 있다.

   즉, **방어코드**가 필요.

3. ES6에서 도입된 **매개변수 기본값**을 사용하면 함수 내 수행하던 인수 체크 및 초기화 가능

   [예제 26-59]

   ```
   function sum ( x = 0, y = 0 ) {
       return x + y;
   }

   console.log(sum(1, 2));  // 3
   console.log(sum(1));     // 1
   ```

4. Rest 파라미터에는 기본값을 지정할 수 없다.

5. 매개변수 기본값은 length 프로퍼티와 arguments 객체에 아무런 영향 X